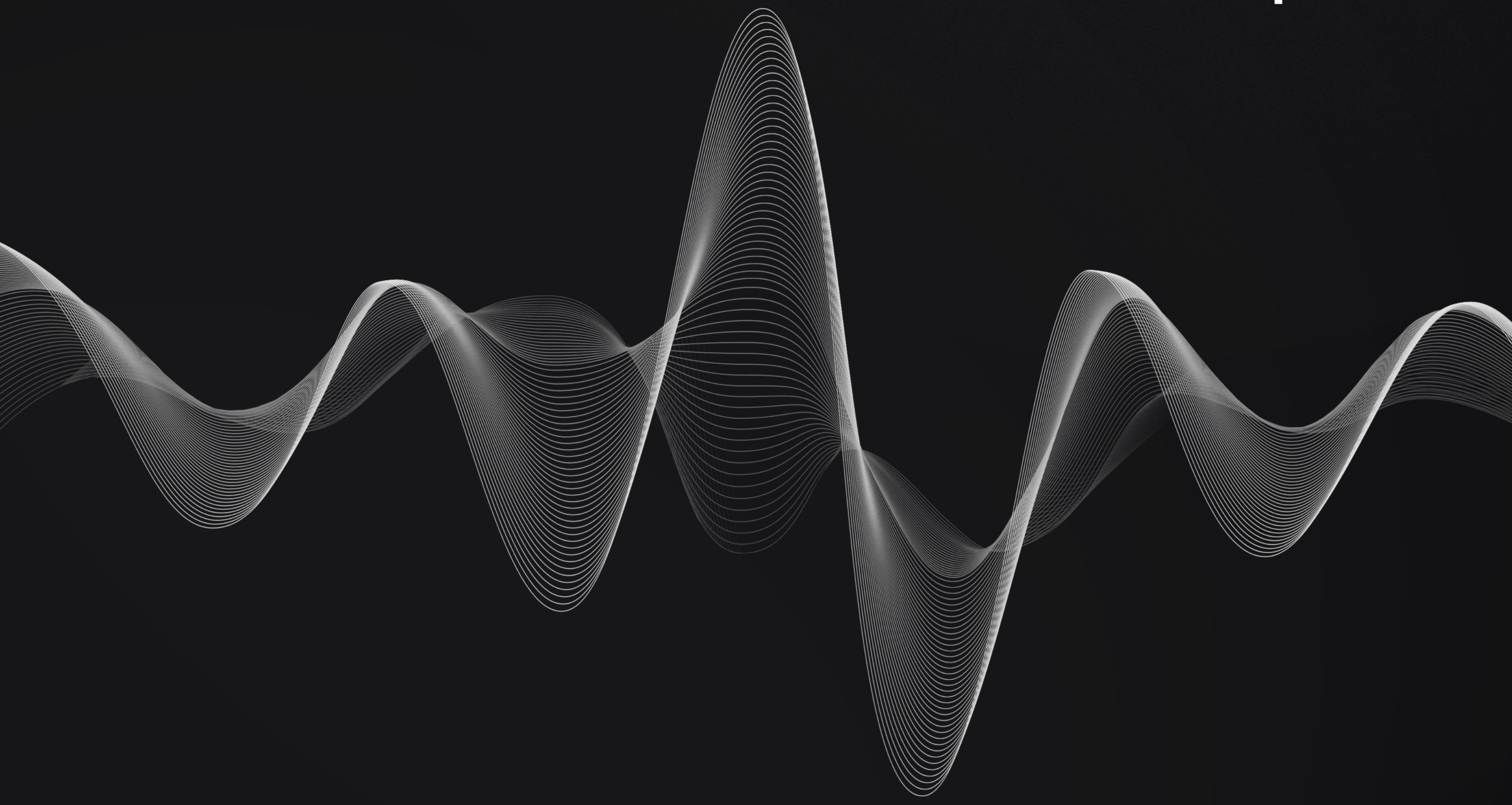
RESONANCE

NettyWorth

NettyWorth

NettyWorth Smart Contracts Audit Report



Blockchain, Emerging Technology, and Web2
CYBERSECURITY PRODUCT & SERVICE ADVISORY

Document Control

PUBLIC **FINAL**(v2.2)

Audit_Report_NTYW-LEN_FINAL_22

Oct 14, 2024	○	v0.1	Luis Arroyo: Initial draft
Oct 14, 2024	○	v0.2	Luis Arroyo: Added findings
Oct 15, 2024	🟢	v1.0	Charles Dray: Approved
Oct 18, 2024	○	v1.1	Luis Arroyo: Added findings
Oct 22, 2024	○	v1.2	Luis Arroyo: Reviewed findings
Oct 23, 2024	🟡	v2.0	Charles Dray: Finalized
Nov 25, 2024	○	v2.1	Luis Arroyo: Reviewed findings
Nov 25, 2024	○	v2.2	Charles Dray: Published

Points of Contact	Ivan Ferrera	NettyWorth	ivan@nettyart.io
	Charles Dray	Resonance	charles@resonance.security
Testing Team	Luis Arroyo	Resonance	luis.arroyo@resonance.security
	João Simões	Resonance	joao@resonance.security
	Michał Bazyli	Resonance	michal@resonance.security
	Ilan Abitbol	Resonance	ilan@resonance.security

Copyright and Disclaimer

Contents

1 Document Control 2

Copyright and Disclaimer 2

2 Executive Summary 4

System Overview 4

Repository Coverage and Quality 4

3 Target 5

4 Methodology 6

Severity Rating 7

Repository Coverage and Quality Rating 8

5 Findings 9

Collateral NFTs Can Be Drained By Unauthorized User 11

Transfer Receipts Not Using The Correct Functions Blocks Payback And Close Loans 12

Loans Cannot Be Liquidated If Promissory Receipt Is Transferred 13

Execution At Deadlines Should Be Allowed 14

Default Loans Could Still Be Paid Back By Borrowers 15

Centralization Risk For Trusted Owners 16

Unsafe Downcast 17

Loans With High APR Cannot Be Paid Back 18

adminFee Can Be Set To 100% 19

abi.encodePacked() Should Not Be Used With Dynamic Types When Passing The Result To A Hash Function Such As keccak256() 20

Incorrect Calculation Of Array Sizes For Allowed/Blocked Collection And Tokens 21

Potential Interest Calculation Precision Issues 22

Possible Scenario For Signature Replay Attack 23

New Deployment Causes Loss Of Information About Used Nonces 24

Critical Changes Should Use Two-step Procedure 25

Consider Disabling renounceOwnership() 26

Events That Mark Critical Parameter Changes Should Contain Both The Old And The New Value 27

Non-external Function Names (Private Or Internal) Should Begin With An Underscore 28

Non-external Variable (Private Or Internal) Should Begin With An Underscore 29

++i/i++ Should Be unchecked{++i}/unchecked{i++} 30

A Proof of Concepts 31

Executive Summary

NettyWorth contracted the services of Resonance to conduct a comprehensive security audit of their smart contracts between October 8, 2024 and October 15, 2024. The primary objective of the assessment was to identify any potential security vulnerabilities and ensure the correct functioning of smart contract operations.

During the engagement, Resonance allocated 2 engineers to perform the security review. The engineers, including an accomplished professional with extensive proficiency in blockchain and smart-contract security, encompassing specialized skills in advanced penetration testing, and in-depth knowledge of multiple blockchain protocols, devoted 6 days to the project. The project's test targets, overview, and coverage details are available throughout the next sections of the report.

The ultimate goal of the audit was to provide NettyWorth with a detailed summary of the findings, including any identified vulnerabilities, and recommendations to mitigate any discovered risks. The results of the audit are presented in detail further below.

System Overview

The protocol proposed by NettyWorth is a lending protocol that allows a borrower to obtain a loan for a fixed period of time by offering an NFT as collateral. The system comprises components working together to facilitate NFT-backed loans, providing a seamless experience for borrowers and lenders, such as, a vault, a loan manager, an NFT receipts manager, and a proxy that ties them all together.

The protocol collects the NFT which will be returned to its owner once the loan is paid back, including interest. In case the loan duration ends, the lender can close the loan and obtain the NFT that was sent to the contract as collateral. The tokens and NFTs used for the loans are controlled via an allowed tokens list.

Repository Coverage and Quality

This section of the report has been redacted by the request of the customer.

Target

The objective of this project is to conduct a comprehensive review and security analysis of the smart contracts that are contained within the specified repository.

The following items are included as targets of the security assessment:

- Repository: `nettyworth/lending-protocol/src`
- Hash: 082167c96d1001122ea268c524e0d1c43414d46c
- Lastest reviewed hash: **65a830b794c68bc57f06c37d4bcc08645804ee66**

The following items are excluded:

- External and standard libraries
- Files pertaining to the deployment process
- Financial-related attack vectors

Methodology

In the context of security audits, Resonance's primary objective is to portray the workflow of a real-world cyber attack against an entity or organization, and document in a report the findings, vulnerabilities, and techniques used by malicious actors. While several approaches can be taken into consideration during the assessment, Resonance's core value comes from the ability to correlate automated and manual analysis of system components and reach a comprehensive understanding and awareness with the customer on security-related issues.

Resonance implements several and extensive verifications based off industry's standards, such as, identification and exploitation of security vulnerabilities both public and proprietary, static and dynamic testing of relevant workflows, adherence and knowledge of security best practices, assurance of system specifications and requirements, and more. Resonance's approach is therefore consistent, credible and essential, for customers to maintain a low degree of risk exposure.

Ultimately, product owners are able to analyze the audit from the perspective of a malicious actor and distinguish where, how, and why security gaps exist in their assets, and mitigate them in a timely fashion.

Source Code Review - Solidity EVM

During source code reviews for Web3 assets, Resonance includes a specific methodology that better attempts to effectively test the system in check:

1. Review specifications, documentation, and functionalities
2. Assert functionalities work as intended and specified
3. Deploy system in test environment and execute deployment processes and tests
4. Perform automated code review with public and proprietary tools
5. Perform manual code review with several experienced engineers
6. Attempt to discover and exploit security-related findings
7. Examine code quality and adherence to development and security best practices
8. Specify concise recommendations and action items
9. Revise mitigating efforts and validate the security of the system

Additionally and specifically for Solidity EVM audits, the following attack scenarios and tests are recreated by Resonance to guarantee the most thorough coverage of the codebase:

- Reentrancy attacks
- Frontrunning attacks
- Unsafe external calls
- Unsafe third party integrations
- Denial of service
- Access control issues

- Inaccurate business logic implementations
- Incorrect gas usage
- Arithmetic issues
- Unsafe callbacks
- Timestamp dependence
- Mishandled panics, errors and exceptions

Severity Rating

Security findings identified by Resonance are rated based on a Severity Rating which is, in turn, calculated off the **impact** and **likelihood** of a related security incident taking place. This rating provides a way to capture the principal characteristics of a finding in these two categories and produce a score reflecting its severity. The score can then be translated into a qualitative representation to help customers properly assess and prioritize their vulnerability management processes.

The **impact** of a finding can be categorized in the following levels:

1. Weak - Inconsequential or minimal damage or loss
2. Medium - Temporary or partial damage or loss
3. Strong - Significant or unrecoverable damage or loss

The **likelihood** of a finding can be categorized in the following levels:

1. Unlikely - Requires substantial knowledge or effort or uncontrollable conditions
2. Likely - Requires technical knowledge or no special conditions
3. Very Likely - Requires trivial knowledge or effort or no conditions

		Likelihood		
		Very Likely	Likely	Unlikely
Impact	Strong	Critical	High	Medium
	Medium	High	Medium	Low
	Weak	Medium	Low	Info

Repository Coverage and Quality Rating

The assessment of Code, Tests, and Documentation coverage and quality is one of many goals of Resonance to maintain a high-level of accountability and excellence in building the Web3 industry. In Resonance it is believed to be paramount that builders start off with a good supporting base, not only development-wise, but also with the different security aspects in mind. A product, well thought out and built right from the start, is inherently a more secure product, and has the potential to be a game-changer for Web3's new generation of blockchains, smart contracts, and dApps.

Accordingly, Resonance implements the evaluation of the code, the tests, and the documentation on a score **from 1 to 10** (1 being the lowest and 10 being the highest) to assess their quality and coverage. In more detail:

- Code should follow development best practices, including usage of known patterns, standard libraries, and language guides. It should be easily readable throughout its structure, completed with relevant comments, and make use of the latest stable version components, which most of the times are naturally more secure.
- Tests should always be included to assess both technical and functional requirements of the system. Unit testing alone does not provide sufficient knowledge about the correct functioning of the code. Integration tests are often where most security issues are found, and should always be included. Furthermore, the tests should cover the entirety of the codebase, making sure no line of code is left unchecked.
- Documentation should provide sufficient knowledge for the users of the system. It is useful for developers and power-users to understand the technical and specification details behind each section of the code, as well as, regular users who need to discern the different functional workflows to interact with the system.

Findings

During the security audit, several findings were identified to possess a certain degree of security-related weaknesses. These findings, represented by unique IDs, are detailed in this section with relevant information including Severity, Category, Status, Code Section, Description, and Recommendation. Further extensive information may be included in corresponding appendices should it be required.

An overview of all the identified findings is outlined in the table below, where they are sorted by Severity and include a **Remediation Priority** metric asserted by Resonance's Testing Team. This metric characterizes findings as follows:

- ▁ **"Quick Win"** Requires little work for a high impact on risk reduction.
- ▄ **"Standard Fix"** Requires an average amount of work to fully reduce the risk.
- █ **"Heavy Project"** Requires extensive work for a low impact on risk reduction.

RES-01	🟪	Collateral NFTs Can Be Drained By Unauthorized User	▄	Resolved
RES-02	🟪	Transfer Receipts Not Using The Correct Functions Blocks Payback And Close Loans	█	Resolved
RES-03	🟪	Loans Cannot Be Liquidated If Promissory Receipt Is Transferred	▄	Resolved
RES-04	🟥	Execution At Deadlines Should Be Allowed	▁	Resolved
RES-05	🟥	Default Loans Could Still Be Paid Back By Borrowers	▄	Resolved
RES-06	🟧	Centralization Risk For Trusted Owners	█	Acknowledged
RES-07	🟧	Unsafe Downcast	▄	Resolved
RES-08	🟧	Loans With High APR Cannot Be Paid Back	▁	Resolved
RES-09	🟩	adminFee Can Be Set To 100%	▁	Resolved
RES-10	🟩	abi.encodePacked() Should Not Be Used With Dynamic Types When Passing The Result To A Hash Function Such As keccak256()	▁	Resolved
RES-11	🟩	Incorrect Calculation Of Array Sizes For Allowed/Blocked Collection And Tokens	▁	Resolved
RES-12	🟩	Potential Interest Calculation Precision Issues	▄	Resolved

RES-13	Possible Scenario For Signature Replay Attack		Resolved
RES-14	New Deployment Causes Loss Of Information About Used Nonces		Resolved
RES-15	Critical Changes Should Use Two-step Procedure		Resolved
RES-16	Consider Disabling renounceOwnership()		Resolved
RES-17	Events That Mark Critical Parameter Changes Should Contain Both The Old And The New Value		Resolved
RES-18	Non-external Function Names (Private Or Internal) Should Begin With An Underscore		Resolved
RES-19	Non-external Variable (Private Or Internal) Should Begin With An Underscore		Resolved
RES-20	++i/i++ Should Be unchecked{++i}/unchecked{i++}		Resolved

Collateral NFTs Can Be Drained By Unauthorized User

Critical **RES-NTYW-LEN01** Access Control **Resolved**

Code Section

- `contracts/CryptoVault.sol#L42`

Description

After a loan is created, a NFT that is stored as collateral in the Vault can only be obtained by the lender or borrower after the loan is finished or liquidated.

However, it is possible for an unauthorized user to obtain all the nft from the vault. This can be done because the `safeBatchTransfer()` function allows any user to call it as it is defined as `public`.

Recommendation

It is recommended to modify the `safeBatchTransfer()` function so it can only be called by the proxy or internally within the contract.

Status

The issue has been fixed in commit 7fd382c6ad10ce9b693fa1a707bc8f15e9117c00.

Transfer Receipts Not Using The Correct Functions Blocks Payback And Close Loans

Critical **RES-NTYW-LEN02** Business Logic **Resolved**

Code Section

- `contracts/NettyWorthProxy.sol#L408`
- `contracts/NettyWorthProxy.sol#L421`

Description

`PromissoryReceipt` and `ObligationReceipt` can be transferred using the basic functions for transferring NFTs such as `safeTransferFrom()` or `transferFrom()` but this will cause a problem in the protocol as it does not update the borrower or lender fields in the `LoanManager` contract.

This prevents liquidation or repayment of a loan whose any of its receipts have been transferred by using any of the methods mentioned above.

Recommendation

In order to allow the use of these functions when transferring receipts, it is necessary to override the functions to take into account the changes in the LoanManager contract instead of leaving the default functions.

Status

The issue has been fixed in commit 7fd382c6ad10ce9b693fa1a707bc8f15e9117c00.

Loans Cannot Be Liquidated If Promissory Receipt Is Transferred

Critical **RES-NTYW-LEN03** Business Logic **Resolved**

Code Section

- `contracts/NettyWorthProxy.sol#L408`
- `contracts/CryptoVault.sol#L151`

Description

`PromissoryReceipt` can be transferred using the `transferPromissoryReceipt()` function but this will cause a problem in the protocol as the `updateLender()` function from `LoanManager` does not update the `_assetsHash` neither `_assets` maps from the `CryptoVault` contract.

This prevents liquidation of a loan whose promissory receipt has been transferred by using the correct `transferPromissoryReceipt()` function.

Recommendation

To perform the lender change correctly, it is recommended to update the mentioned fields after the receipt transfer. to do so, besides updating the new lender in the `loanManager` contract, it is also necessary to update the `_assets` and `_assetsHash` of the `cryptovault` contract to correlate the collateral with the new lender.

Status

The issue has been fixed in commit 7fd382c6ad10ce9b693fa1a707bc8f15e9117c00.

Execution At Deadlines Should Be Allowed

High **RES-NTYW-LEN04** Timestamp Dependence **Resolved**

Code Section

- `contracts/NettyWorthProxy.sol#L361`

Description

When a loan is created, both parties agree on the loan duration. The protocol implements mechanisms to avoid lenders to liquidate the loan before it ends but it is still possible for lenders to liquidate a loan in the last block of the loan period.

The `forCloseLoan()` function should check if the `block.timestamp` is greater than `loan.loanDuration` instead of checking if it is greater or equal.

This may result in borrowers potentially losing the collateral NFT at the last timestamp of the loan's duration.

Recommendation

It is recommended to update the duration check to ensure that lenders cannot liquidate positions in the last block, e.g. by checking that the timestamp is greater than the actual duration.

Status

The issue has been fixed in commit d6f72ef6555ea24ba4a1d227acd773b4115a76b5.

Default Loans Could Still Be Paid Back By Borrowers

High | RES-NTYW-LEN05 | Business Logic | **Resolved**

Code Section

- `contracts/NettyWorthProxy.sol`

Description

The protocol allows a loan that has been finalized to be repaid as long as it is not liquidated. This makes the interest and the fees incorrect.

The borrower could wait until lender calls `forCloseLoan()` function to frontrun him and close the loan by calling `payBackLoan()` and obtain a loan for a longer period than initially stated.

Recommendation

It is recommended to add a check that allows repayment of a loan while it is within the time frame established at the beginning of the loan.

Status

The issue has been fixed in commit 14d57ea447cfb32bc58fb15b5799a39fbaa98657.

Centralization Risk For Trusted Owners

Medium **RES-NTYW-LEN06** Access Control **Acknowledged**

Code Section

- `contracts/LoanManager.sol`
- `contracts/CryptoVault.sol`
- `contracts/LoanReceipt.sol`
- `contracts/LoanWhiteListCollection.sol`
- `contracts/NettyWorthProxy.sol`

Description

The protocol contains several functions access controlled by owners with privileged rights in charge of performing admin tasks like setting the proxy or setting fees. These owners need to be trusted to not perform malicious updates on the protocol.

Recommendation

It is recommended to implement solutions like a multisignature wallet to distribute admin control among multiple trusted parties. This ensures that critical actions can only be executed if a predefined quorum of trusted parties approves the action, reducing the risk of unilateral decisions or key compromise.

Status

The issue was acknowledged by NettyWorth's team. The development team stated that they will implement the multi sig wallet to mitigate the risk.

Unsafe Downcast

Medium **RES-NTYW-LEN07** Data Validation **Resolved**

Code Section

- `contracts/LoanManager.sol#L132`

Description

When a type is downcast to a smaller type, the higher order bits are truncated, effectively applying a modulo to the original value. Without any other checks, this wrapping will lead to unexpected behavior and bugs.

In this protocol, failures caused by collisions could lead to errors in loan creation.

Recommendation

To mitigate the risk of this insecure downcasting that leads to hash collisions, it is recommended to avoid unnecessary downcasting and use `uint256`.

Status

The issue has been fixed in commit d6f72ef6555ea24ba4a1d227acd773b4115a76b5.

Loans With High APR Cannot Be Paid Back

Medium **RES-NTYW-LEN08** Data Validation **Resolved**

Code Section

- `contracts/LoanManager.sol#L147`

Description

The `createLoan()` function allows the creation of a loan with an `_aprBasisPoints` higher than 100%.

When a borrower tries to payback a loan with this huge rate, the require that checks the loan.`aprBasisPoints` in the `getPayoffAmount()` function will revert.

Recommendation

It is recommended to add the necessary requires in the `createLoan()` function to check that the rate is correct.

Status

The issue has been fixed in commit d6f72ef6555ea24ba4a1d227acd773b4115a76b5.

adminFee Can Be Set To 100%

Low RES-NTYW-LEN09 Data Validation **Resolved**

Code Section

- `contracts/NettyWorthProxy.sol#L71-L78`

Description

The `updateAdminFee()` function allows an owner to set a fee of 100% which may not be practical or could be considered predatory lending.

Recommendation

It is recommended to limit the maximum fee to an appropriate value.

Status

The issue has been fixed in commit d6f72ef6555ea24ba4a1d227acd773b4115a76b5.

abi.encodePacked() Should Not Be Used With Dynamic Types When Passing The Result To A Hash Function Such As keccak256()

Low **RES-NTYW-LEN10** Data Validation **Resolved**

Code Section

- `contracts/LoanManager.sol#L133`

Description

Unless there is a compelling reason, `abi.encode()` should be preferred instead of `abi.encodePacked()`.

`abi.encode()` will pad parameters to 32 bytes, which will prevent hash collisions.

For example, `abi.encodePacked(0x123,0x456)` returns `0x123456` which will collide with `abi.encodePacked(0x1,0x23456)`. but `abi.encode(0x123,0x456)` will result in `0x0...1230...456`

If there is only one argument to `abi.encodePacked()` it can often be cast to `bytes()` or `bytes32()` instead.

If all arguments are strings and or bytes, `bytes.concat()` should be used instead.

It should be noted that the severity of this finding has been decreased since no dynamic types are being used inside the function abi.encodePacked().

Recommendation

It is recommended to change `abi.encodePacked()` to `abi.encode()` to add an extra layer in collision prevention.

Status

The issue has been fixed in commit d6f72ef6555ea24ba4a1d227acd773b4115a76b5.

Incorrect Calculation Of Array Sizes For Allowed/Blocked Collection And Tokens

Low | **RES-NTYW-LEN11** | Arithmetic Issues | **Resolved**

Code Section

- `contracts/WhiteListCollection.sol#L26`
- `contracts/WhiteListCollection.sol#L40`
- `contracts/WhiteListCollection.sol#L55`
- `contracts/WhiteListCollection.sol#L71`

Description

When an owner wants to add/remove collections or allowed tokens, he calls the `whitelistCollection` contract functions with the array of addresses he wants to add/remove.

In case this array has only one element, the calculation of the collection/tokens total will be wrong, since it subtracts 1 from the array size (which will be 1) and adds it to the total, meaning no increment or decrement.

Recommendation

It is recommended to fix the calculation of the collection size when updating the `totalWhitelistCollection` or the `totalWhitelistErc20Token` variables.

Status

The issue has been fixed in commit d6f72ef6555ea24ba4a1d227acd773b4115a76b5.

Potential Interest Calculation Precision Issues

Low **RES-NTYW-LEN12** Arithmetic Issues **Resolved**

Code Section

- `contracts/LoanManager.sol#L152`

Description

In Solidity, dividing integers truncates the result to the nearest smallest integer, meaning that any fractional part of the result is discarded.

The function `getPayoffAmount()` multiplies three values: `loanAmount`, `aprBasisPoints`, and `loanDurationinSeconds` and then, the result is divided by `(BPS * SECONDS_IN_YEAR)`.

This division might result in a loss of precision in cases where the product of `(loanAmount * aprBasisPoints * loanDurationinSeconds)` is not perfectly divisible by and is less than `(BPS * SECONDS_IN_YEAR)`. This truncation could lead to small differences in the calculated interest, particularly for loans with:

- Short durations
- Small loan amounts
- Low APR values

Recommendation

To avoid precision loss when dividing, it is recommended to use a scaling factor to simulate floating-point arithmetic. By scaling up intermediate values, it is possible to keep precision and minimize the impact of truncation.

Status

The issue has been fixed in commit d6f72ef6555ea24ba4a1d227acd773b4115a76b5.

Possible Scenario For Signature Replay Attack

Low **RES-NTYW-LEN13** Data Validation **Resolved**

Code Section

- `contracts/library/SignatureUtils.sol#L52`

Description

The `validateNftDepositSignature()` function validates the deposit of an NFT. This function does not use any nonce, allowing a malicious user to replay the signature.

This function is not currently used in the protocol, so exploitation is not possible. However, when it starts being used, this vulnerability would arise and negatively affect the protocol.

Recommendation

Consider including a `nonce` in the signature message and parameter, similar to other `validate()` functions.

Status

The issue has been fixed in commit d6f72ef6555ea24ba4a1d227acd773b4115a76b5.

New Deployment Causes Loss Of Information About Used Nonces

Info **RES-NTYW-LEN14** Business Logic **Resolved**

Code Section

- `contracts/NettyWorthProxy.sol#L473-L474`

Description

During loan creation, the protocol calls the `SignatureUtils` contract that manages and validates the signatures of the lender and borrower.

The functions used for these validations use a nonce that is previously validated and stored in the `_nonceUsedForUser` mapping from the proxy, so it can only be used once per loan.

In case that a new version of the contract is deployed, the `_nonceUsedForUser` mapping will be empty and the data in the old contract will not be accessible, allowing the reuse of nonces and the replay of signatures.

Recommendation

It is recommended to store the nonces used by the users in another contract different from the proxy. this way, the proxy will avoid overwriting the mapping when it is updated, allowing replay attacks, since the used nonce will be available again.

Status

The issue has been fixed in commit d6f72ef6555ea24ba4a1d227acd773b4115a76b5.

Critical Changes Should Use Two-step Procedure

Info **RES-NTYW-LEN15** Data Validation **Resolved**

Code Section

- `contracts/LoanManager.sol`
- `contracts/CryptoVault.sol`
- `contracts/LoanReceipt.sol`
- `contracts/NettyWorthProxy.sol`

Description

The critical procedures used to change proxies, fees or owners in this protocol should use a two step process in order to ensure the modification is indeed intended.

Recommendation

The lack of two-step procedures for critical operations leaves them prone to mistakes. Consider adding two-step procedures on critical functions, such as, `setProxyManager()`, `setOpen()` or `setAdminWallet()`.

Status

The issue has been fixed in commit d6f72ef6555ea24ba4a1d227acd773b4115a76b5.

Consider Disabling renounceOwnership()

Info **RES-NTYW-LEN16** Code Quality **Resolved**

Code Section

- `contracts/LoanManager.sol`
- `contracts/CryptoVault.sol`
- `contracts/LoanReceipt.sol`
- `contracts/LoanWhiteListCollection.sol`
- `contracts/NettyWorthProxy.sol`

Description

As NettyWorth does not include eventually giving up all ownership control, consider overwriting OpenZeppelin's Ownable's `renounceOwnership()` function in order to disable it.

This will prevent the protocol from being left unattended in the event that the owners renounce ownership.

Recommendation

It is recommended to disable the function `renounceOwnership()` by overriding it.

Status

The issue has been fixed in commit d6f72ef6555ea24ba4a1d227acd773b4115a76b5.

Events That Mark Critical Parameter Changes Should Contain Both The Old And The New Value

Info **RES-NTYW-LEN17** Code Quality **Resolved**

Code Section

- `contracts/NettyWorthProxy.sol`

Description

`UpdatedAdminWallet()` and `UpdatedAdminFee()` events should emit the previous information and the new one to improve traceability.

This should especially be done if the new value is not required to be different from the old value

Recommendation

Consider adding the old parameter to the events `UpdatedAdminWallet()` and `UpdatedAdminFee()`.

Status

The issue has been fixed in commit d6f72ef6555ea24ba4a1d227acd773b4115a76b5.

Non-external Function Names (Private Or Internal) Should Begin With An Underscore

Info **RES-NTYW-LEN18** Code Quality **Resolved**

Code Section

- `contracts/LoanReceipt.sol`
- `contracts/library/SignatureUtils.sol`

Description

According to the Solidity Style Guide, non-external function names should begin with an underscore.

Recommendation

Consider renaming the affected private functions.

Status

The issue has been fixed in commit d6f72ef6555ea24ba4a1d227acd773b4115a76b5.

Non-external Variable (Private Or Internal) Should Begin With An Underscore

Info **RES-NTYW-LEN19** Code Quality **Resolved**

Code Section

- `contracts/LoanManager.sol`
- `contracts/CryptoVault.sol`

Description

According to the Solidity Style Guide, non-external variable names should begin with an underscore.

Recommendation

Consider renaming the affected private variables.

Status

The issue has been fixed in commit d6f72ef6555ea24ba4a1d227acd773b4115a76b5.

++i/i++ Should Be unchecked{++i}/unchecked{i++}

Info **RES-NTYW-LEN20** Gas Optimization **Resolved**

Code Section

- `contracts/WhiteListCollection.sol`

Description

When it is not possible for counters to overflow, as is the case when used in for and while-loops, they should be `unchecked`. The `unchecked` keyword is new in solidity version 0.8.0, so this only applies to that version or higher, which these instances are. This saves 30-40 gas per loop.

Recommendation

Consider increment the counters in `unchecked` code in order to save gas.

Status

The issue has been fixed in commit 14d57ea447cfb32bc58fb15b5799a39fbaa98657.

Proof of Concepts

RES-01 Collateral NFTs Can Be Drained By Unauthorized User

```
function testGetCollaterals() public {
    testAcceptLoan();

    assertEq(nft.ownerOf(1), address(cryptoVault));

    uint256[] memory tokenIds = new uint256[](1);
    tokenIds[0] = 1;

    vm.prank(attacker);
    cryptoVault.safeBatchTransfer(
        nft,
        address(cryptoVault),
        attacker,
        tokenIds
    );

    assertEq(nft.ownerOf(1), attacker);
}
```

RES-02 Transfer Receipts Not Using The Correct Functions Blocks Payback And Close Loans

```
function testTransferAndClose() public {
    testAcceptLoan();
    uint256 loanId =
↪   100581929920708749893525838127504967854099106449984045834507925510301982038586;

    vm.prank(lender1);
    lenderReceipt.safeTransferFrom(address(lender1), lender2, 1);

    vm.warp(11 days);

    vm.prank(lender2);
    bool success = netty.forCloseLoan(loanId);
    assertTrue(success);
}

////////////////////////

function testTransferAndPayback() public {
    testAcceptLoan();
    uint256 loanId =
↪   100581929920708749893525838127504967854099106449984045834507925510301982038586;

    vm.prank(borrower1);
    borrowerReceipt.safeTransferFrom(address(borrower1), borrower2, 1);
```

```
        vm.prank(borrower2);
        bool success = netty.payBackLoan(loanId, address(token));
        assertTrue(success);
    }
```

RES-03 Loans Cannot Be Liquidated If Promissory Receipt Is Transferred

```
    function testTransferOkAndClose() public {
        testAcceptLoan();
        uint256 loanId =
↪   100581929920708749893525838127504967854099106449984045834507925510301982038586;

        vm.startPrank(lender1);
        lenderReceipt.setApprovalForAll(address(netty), true);
        netty.transferPromissoryReceipt(loanId, 1, lender2);
        vm.stopPrank();

        vm.warp(11 days);

        vm.prank(lender2);
        bool success = netty.forCloseLoan(loanId);
        assertTrue(success);
    }
```

RES-04 Execution At Deadlines Should Be Allowed

```
function test_defaultValidLoan() public {

        vm.startPrank(lender);

        // Set up a LoanRequest
        SignatureUtils.LoanRequest memory request = SignatureUtils.LoanRequest({
            tokenId: 1,
            nftContractAddress: address(nft),
            erc20TokenAddress: address(token),
            borrower: borrowerS,
            loanAmount: 1e18,
            aprBasisPoints: 500,
            loanDuration: block.timestamp + 10 days,
            nonce: 1
        });

        // Generate a valid signature for this request
        bytes32 hash = keccak256(
            abi.encode(
                request.tokenId,
                request.nftContractAddress,
                request.erc20TokenAddress,
                request.borrower,
                request.loanAmount,
                request.aprBasisPoints,
                request.loanDuration,
                request.nonce
            )
        );

        bytes memory signature = signMessage(
            keccak256(
                abi.encodePacked("\x19Ethereum Signed Message:\n32", hash)
            )
        );

        // Accept the loan request
        (uint256 receiptIdBorrower, uint256 receiptIdLender) = proxy
            .acceptLoanRequest(signature, request);

        vm.stopPrank();

        // Forward to the day the loan ends
        vm.warp(block.timestamp + 10 days);

        // Lender can close the loan in the last day
        vm.prank(lender);
        proxy.forCloseLoan(14772483896491979347);
}
```

RES-05 Default Loans Could Still Be Paid Back By Borrowers

```
    function test_returnLoanAfterDuration() public {

        vm.startPrank(lender);

        // Set up a LoanRequest
        SignatureUtils.LoanRequest memory request = SignatureUtils.LoanRequest({
            tokenId: 1,
            nftContractAddress: address(nft),
            erc20TokenAddress: address(token),
            borrower: borrowerS,
            loanAmount: 1e18,
            aprBasisPoints: 500,
            loanDuration: block.timestamp + 10 days,
            nonce: 1
        });

        // Generate a valid signature for this request
        bytes32 hash = keccak256(
            abi.encode(
                request.tokenId,
                request.nftContractAddress,
                request.erc20TokenAddress,
                request.borrower,
                request.loanAmount,
                request.aprBasisPoints,
                request.loanDuration,
                request.nonce
            )
        );

        bytes memory signature = signMessage(
            keccak256(
                abi.encodePacked("\x19Ethereum Signed Message:\n32", hash)
            )
        );

        // Accept a loan request
        (uint256 receiptIdBorrower, uint256 receiptIdLender) = proxy
            .acceptLoanRequest(signature, request);

        // Forward more time than duration
        vm.stopPrank();
        vm.warp(block.timestamp + 20 days);

        // Borrower can payback the loan
        vm.startPrank(borrowerS);
        proxy.payBackLoan(14772483896491979347, address(token));
}
```